Organization and management of ATLAS offline software releases

E. Obreshkov, S. Albrand, J. Collot, J. Fulachier, F. Lambert, C. Adam-Bourdarios, C. Arnault, V. Garonne, D. Rousseau, A. Schaffer, et al.

ORGANIZATION AND MANAGEMENT OF ATLAS SOFTWARE RELEASES

E. Obreshkov, INRNE, Bulgarian Academy of Sciences, Bulgaria.
S. Albrand, J. Collot, J. Fulachier, F. Lambert, LPSC, IN2P3/CNRS, Grenoble, France.
C. Adam-Bourdarios, C. Arnault, V. Garonne, D. Rousseau, A. Schaffer,
LAL;IN2P3-CNRS, Univ Paris-Sud 11; Orsay, France.
H. von der Schmitt, Max-Planck-Institut für Physik, München, Germany.
A. De Salvo, INFN - Roma1, Italy.
V. Kabachenko, Institute for High Energy Physics (IHEP), Protvino, Russia.
Z. Ren, D. Qing, Institute of Physics Academia Sinica, Taiwan.
E. Nzuobontane, P. Sherwood, B. Simmons, University College, London, UK.
S. George, G. Rybkine, Royal Holloway, University of London, UK.
S. Lloyd, Queen Mary, University of London, UK.
A.Undrus, BNL, Upton, NY, USA.
S. Youssef, Boston University, MA, USA.
D. Quarrie, LBNL, CA, USA.
T. Hansl-Kozanecka, UCSC, USA and CEA/DAPNIA, France.
F. Luehring, University of Indiana, USA.
E. Moyse, University of Massachusetts, Amherst, MA, USA.
S. Goldfarb, University of Michigan, USA.

Abstract

ATLAS is one of the largest collaborations ever undertaken in the physical sciences. This paper explains how the software infrastructure is organized to manage collaborative code development by around 300 developers with varying degrees of expertise and situated in 30 different countries. ATLAS offline software currently consists of about 2 million source lines of code contained in 6800 C++ classes, organized in almost 1000 packages. We will describe how releases of the offline ATLAS software are built, validated and subsequently deployed to remote sites. Several software management tools have been used, the majority of which are not ATLAS specific; we will show how they have been integrated.

INTRODUCTION

ATLAS is one of the largest collaborative efforts ever undertaken in the physical sciences. About 2000 physicists participate, from more than 150 universities and laboratories in more than 30 countries.

The software is correspondingly large both in size, and also in the number of developers involved. These considerations have lead us to put into place policies and technical methods to make the development and release mechanism as smooth and efficient as possible.

A release of ATLAS offline software contains approximately 1000 code directories containing about 2 MSLOC[1]. The code is almost exclusively in C++ although some legacy FORTRAN code remains. Job options files for run time control of code execution are written in Python.

Currently about 300 members of the collaboration are registered as involved in development, although of course many of them are not engaged in this activity full-time. The number of developers is expected to increase as we move towards real data taking, as people previously working on building the detector will become available to work on software.

On the other hand, we expect the total size of the release to be reduced, as this is normal in the development process as software reaches maturity.

Since the first release of ATLAS offline software for the Physics Technical Design Report in May 2000, 12 major releases of the software have been produced.

By 2001, it became clear that the difficulties involved in managing software produced by a large number of people who are widely distributed geographically and in their majority not expert software engineers, required the establishment of a management structure, and a coordinated set of tools. The ATLAS software infrastructure team was put into place to provide the technical support necessary. This

[1] MSLOC 10^6 Source Lines of Code

paper explains the organization we have put in place, the different tools we use, and how they work together.

RELEASE ORGANIZATION

The following parties are involved in the release in either a management or a technical capacity:

- The Chief Architect directs the software effort and takes the final decisions on the software design and the schedule for the introduction of new features of the software or the release structure.
- The Software Project Management Board (SPMB) approves major policy decisions in consensus with the Chief Architect and serves as a forum for discussion of policy. All interested communities, such as the ATLAS detector groups, the testbeam group, the physics working groups and the core software group have representatives on the SPMB.
- The Software Infrastructure Team (SIT) is the technical group, which brings together the providers and the users of the tools used to build the software and its distribution kit. The group meets every two weeks to discuss coordination issues and schedules.
- The Software Librarians are the principle users of the build tools; they actually build the software releases and oversee distribution of the software.
- The Release Coordinators have a policing role. They must validate that software submitted by each community functions properly and meets the design goals of each release. They can ask for changes in the submitted code or refuse to accept code. They decide when a release is ready to be built. As the role of release coordinator involves a heavy responsibility this position is periodically rotated.
- The Package Managers and Package Developers manage and write the software that we produce. For smaller packages a single person holds both of these roles. Larger packages have a hierarchical organization whereby new code written by developers must be approved by a manger before inclusion in the release.

RELEASE POLICIES AND THE RELEASE CYCLE

ATLAS uses several kinds of builds in order to make a release. As the date of releases approaches, it becomes more and more difficult to insert new packages, or new versions of packages in the software. Indeed, we have a mechanism to lock parts of the release, which prevents such an insertion. Figure 1 shows the chronology of these different releases, and introduces two of the tools we use: Tag Collector [1], which is a database application that holds the list of the package versions which make up the release, and contains some locking functions, and NICOS [2], which runs the nightly builds.

Every 24 hours there is a nightly build of the software. Usually the “nightly” is an incremental build (to save time), but once a week, it is a full build of all packages (~20 hours). Developers submit new versions of software to the "nightlies" via the Tag Collector. Nightly builds then test the new versions and the integration of the different packages.

Every 3-4 weeks there is a development or minor release of the software. New code is expected to follow the software design but developers do have considerable freedom in what code they submit. These builds provide a stable version to develop against, but are not rigorous production quality releases – they are not validated for use for physics.

Every 6 months there is a production candidate or major release, which provides the basis of the software used in the next production. Restrictions are put on what code can be added to the production release.

In practice we are obliged to follow each production candidate release by several bug fix releases, which fix problems found during testing and validation of the production candidate release and the succeeding bug fix releases. For these bug fix releases, very strict rules for adding code are followed.

THE DEVELOPMENT PROCESS

This section describes the development cycle and tools used by ATLAS to manage the software development process.

Submitting a new version of a package

Using the configuration management tool CMT [3], a developer checks out a package from the CVS repository and works on it, building against an existing release. When he is satisfied with his changes, the new code version is committed to CVS, and given an identifying name (a “tag”). If the developer wants the new code version to be included in a release, the tag must be entered into the Tag Collector

database. The developer uses the Tag Collector to add his new tag to one of the releases that are open for tag collection. If a production release is pending, tag collection for the production release may be locked making it impossible for the developer to add his new tag. In this case, the developer must request that the release coordinator add the tag to the production release.

Building and distributing the release

Both the nightly builds built by the automatic NICOS system, and the numbered releases built by the librarian use pure CMT commands to make a release. There are minor technical differences in the method for deriving the list of tags used in building the nightly and numbered (development, production, and bug-fix) releases. However both types of builds do build the same set of code containing ("leaf") packages.

Once a software release suitable for production has been built and undergone preliminary validation, the librarian prepares the distribution kit using a suite of shell and python scripts developed for this purpose [4]. The kit is placed in a disk cache accessible via http. Grid site managers and individuals use Pacman [5] to fetch and install the ATLAS software. The kit includes a validation suite (Kit Validation [6]).

Automatic Documentation

Doxygen documentation is produced for every release, and we have made considerable effort over the past year to improve its quality. A large number of packages now include a "main-page" that provides a written overview of the package and its interfaces. Other documentation initiatives are described below.

Quality Assurance and Testing

In the NICOS framework the quick unit and integration tests are run after the release builds. These simple tests prove that the basic release functionalities work and can serve as a prerequisite for more detailed tests. The Run Time Tester (RTT) [7] is used to make regression tests before the release is distributed generally. Regression tests are a comparison of release output to a known standard output generated with a previous release. We also run some other QA tools over the release. They are described in more detail below.

DETAIL OF THE TOOLS USED BY ATLAS FOR RELEASE MANAGEMENT

Configuration Management

ATLAS uses the tool CMT [3] for configuration management, which supports the decomposition of the offline software into packages, or groups of packages also called "projects". Every package or package group must specify its configuration in a text file (the "requirements file"). The requirements file defines the dependencies on other packages ("use" statements) and command line parameters needed to run tools like "make", Doxygen or kit building scripts. It also describes how to make the release components such as applications or libraries, how to use them, for example, how to find the application job options, and how to apply management actions (build, documentation generation, tests, installation etc). Management actions are broadcast through the software hierarchically following the dependency tree.

Tag Collection

The ATLAS Tag Collector [1] is a database application with a powerful web interface. It provides tools for package developers and managers to submit the version tag of their software package for inclusion in builds of the software, management tools to enforce policy, such as the ability to lock parts of the release thus permitting controlled tag submission, and tools for release building.

Tag Collector users are assigned various roles for each software package. Examples of roles are "package developer" who can add a new package version to a release, and "package manager", a developer with the additional right of declaring a new source file to be part of a release. Another role is the "release coordinator" who is responsible for upholding the management policy for building the release. The librarian has tools to construct the correct CMT requirements file for a package group and to tag package groups automatically in the code repository.

Two web interfaces have been provided. One is designed for power users, and provides global access to all the Tag Collector commands. Most users, however, prefer a simpler interface, which accedes to the command in a more structured fashion. Each user has his "home page", which is displayed on logging on. This page is determined according to the declared responsibilities of the user. As the user navigates through the Tag Collector pages, a list of the commands, which are authorized, is provided.

The Tag Collector also provides a web service to execute tag collector commands. Web service commands can be incorporated into scripts by power users and by cooperating applications. Both CMT and NICOS use the web service interface to obtain the list of package versions that must be built together.

The Tag Collector architecture uses an abstract interface to decouple it from CVS and CMT. In this way ATLAS is free to use alternative tools, and Tag Collector can be adapted for other experiments.

Tag Collector can manage several open releases simultaneously. The only restriction is that the parent release of any open release must be declared closed to further changes before the new release is open. ATLAS uses this feature to dedicate one release to one particular specialized set of goals, while allowing another path of development to remain open. Recently this notion was extended to allow "project builds" which divided the software tree into several (~10) projects that can independently be developed and built. For the Tag Collector project builds required the incorporation of rules governing the hierarchy of projects, and extending the web interface to include a graphic representation of the interdependence of projects.

Actions within the tag collector are logged, so that it is possible for a manager to see when a developer assigned a particular package version to a release. Useful statistics are also available: for example one can obtain the total number of leaf packages within a release, and how many of them had a new version with respect to the parent release.

Nightly COntrol System (NICOS)

NICOS manages the nightly builds of the ATLAS software on several different platforms. NICOS uses the ATLAS Tag Collector and CMT to perform the nightly builds. NICOS is designed to make the system flexible and easy to use (see Figure 2):

- Each step within NICOS is modular, can use plug-ins and can be configured individually. Examples of the modular steps are compilation of the code, testing of the compiled code, the analysis of errors generated in compiling and testing the code, and creation of web pages containing a summary of the result of the NICOS run.
- An XML file stores the project configuration of NICOS.
- NICOS supports the projects builds. The projects are built in the order determined by declared projects dependencies.
- NICOS automatically posts information about its progress as it runs, identifies problems, and creates the web pages with build results as shown in Figure 3. The authors of packages that fail to build or pass tests receive automatic e-mail notification making the run results immediately available to the ATLAS developers who are spread over different institutions and countries.
- Build stability is assured by automatic discovery and recovery from build failures. NICOS is configured to run on a local disk system and copy the build onto a distributed file system (currently AFS).

NICOS can be easily configured for any type of software build including the stable releases, work releases with "on-demand" tags and documentation builds (using Doxygen). The XML configuration file consists of mark-up tags corresponding to the steps of the build process. A mark-up tag is followed by the list of NICOS environment variables and commands to be executed in this step.

NICOS provides a framework for tests of different types: quality checks, unit tests, and integration tests. The quality checks and unit test results are posted for each software package on the build summary web page, while the integration test results are summarized separately. About 70 unit and integration tests covering all parts of the software are performed.

ATLAS Testing Nightly (ATN)

ATN [8] is the testing framework based on the QMTest tool [9], which extends functionality of NICOS to integration testing. The major ATN features are:

- The test configuration is stored in XML files. The common format is developed for available for three ATLAS testing frameworks (of which ATN is one).
- ATN automatically finds suites of test descriptions in the software release tree.
- The ATN test results are returned to NICOS for processing and publication.

ATN verifies the test results by checking the exit value returned by each test executable. In addition a search for text patterns of three types is performed:

- "Success" patterns are required in the output of a successful test. The absence of the "success" pattern triggers an error.
- "Warning" and "Error" patterns trigger two levels of alarm.

A default set of typical patterns for ATLAS software tests is provided. Additional patterns can be specified in test configuration files. The packages with warnings or errors in their unit tests as well as failed integration tests are highlighted with different colours on NICOS web pages. ATN sends automatic e-mail problem notifications as specified in the configuration file.

Kit Building and Distribution

The suite of scripts written to build the distribution kit [4] makes extensive use of the tools developed within the GNU Project [10] of the Free Software Foundation [11]. The packaging unit, which was chosen for kit building, follows the project granularity. The scripts package the software into a distribution "kit". Each package of the kit consists of a compressed tar ball and a Pacman file with the package meta-data.

Separate tar-balls include binaries, source code, and header, configuration and data files. This granularity meets the needs of both production and development use cases.

Pacman files reference the packed software and describe how the packages should be fetched, installed, configured and updated. Pacman [5] is used to install, remove and query the software and it aims to maintain the integrity, repeatability and verifiability of installed releases at remote sites.

Quality Assurance Tools

A tool named “checkreq” is run nightly over all packages by NICOS to verify the consistency of the declared package dependencies, and the #include instructions of the C++ code.

RuleChecker, a tool to check compliance with many of the ATLAS C++ coding rules can be run on all or parts of the release source code using RTT. At present this is done on a volunteer basis.

For further information on ATLAS quality assurance see reference [12].

Documentation

In a large collaboration with constantly evolving software it is difficult to provide correct and useful information. A large effort was made in ATLAS to restructure the sources of information and identify the responsibilities for providing and keeping information up-to-date. The task is shared between developers, who are responsible for specific domains (e.g. a sub-detector, or a slice of the software, like reconstruction) and specific people, who provide an overview of the documentation.

Different tools are used at different levels of documentation:

- Standard Web pages [13] provide the oversight and navigation to the different domains. They use a standard layout, which is based on cascading style sheets, and may optionally contain header and sidebar navigation. The content is of broad scope and used by a large number of people.
- A “Workbook”[14] is provided for newcomers to the ATLAS software. It introduces the ATLAS computing environment and provides a consistent set of instructions, which guide the user from the generation of events through reconstruction and basic analysis. It is available on Web and can be printed in PDF format.
- For detailed code documentation Doxygen [15] is used. Doxygen extracts information directly from the ATLAS source code and the included comments. For each package, a “main page” provides introduction and navigation to important classes. The resulting documentation is extracted for each release and is presented in a collapsible tree to the user [16].

The standard Web pages, the Workbook and the Doxygen documentation follow version control and editing rules. A “Wiki” tool called Twiki [17] was chosen for less restricted collaborative writing of documents. It has become very popular; most of the specific and evolving instructions are available there. The “Workbook” is also based on this Twiki, but makes use of an additional Python layer for conversion to PDF [18]. More monitoring and control has become necessary for the Twiki documentation, as proposed in a recent review of the ATLAS Documentation [19].

RECENT IMPROVEMENTS AND FUTURE PLANS

The ATLAS software release is large. A complete build of the ATLAS offline software currently takes ~20 hrs. To improve matters, we have divided the software into “projects”. A project is a group of packages, which can have its own independent development timeline. Higher-level projects can develop against stable versions of lower level ones (closer to the core software). The division of the release into projects was introduced in January 2006; it has required modification of all tools.

A new suite of kit building scripts was developed to construct the distribution kit for each project. Reducing the packaging granularity to that of the projects, the suite has reduced kit-building time significantly. The project kit is sub-divided into platform-dependent (optimized, debug), platform-

independent, source and documentation parts. This allows us to switch to using the distribution kit for CERN installations as well as remote sites. The new scripts are now also used to package some of the ATLAS T/DAQ software projects. The kits also allow the installation of the necessary software in the ATLAS detector control room and in the trigger computing farms.

We expect to see an increasing emphasis on code testing and validation by increasing the use of the Run Time Tester ("RTT") and extending its use to Grid nodes. The RTT is a testing framework that allows automated testing of the ATLAS software on a nightly basis. RTT runs on its own farm of computers and allows lengthy tests of the software.

CONCLUSIONS

ATLAS has developed a suite of tools for code release that works well with our base of about 300 code developers spread out over the world. This suite, combined with our management structure, allows us the flexibility to respond to the inevitable crises in building releases without losing control of the release process.

Most of this suite has been developed and integrated since the advent of the Software Infrastructure Team (SIT). The SIT has taken the critical central role in integrating the tools and using them to make 12 major and a total of over 50 Atlas offline software releases. Approximately 20 people contribute effort to the ATLAS SIT and almost all of the contributors have large responsibilities outside of the SIT. In spite of having multiple responsibilities, these people do dedicate a significant fraction of their time to the SIT and the value of having this dedicated team leading the software releases cannot be overstated.

REFERENCES

1 http://atlastagcollector.in2p3.fr
2 http://www.usatlas.bnl.gov/computing/software/nicos/
3 http://www.cmtsite.org/
4 http://atlas-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/Deployment/
http://atlas-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/PackDist/
5 http://physics.bu.edu/pacman/
6 https://twiki.cern.ch/twiki/bin/view/Atlas/KitValidation
7 http://www.hep.ucl.ac.uk/atlas/AtlasTesting/
8 https://twiki.cern.ch/twiki/bin/view/Atlas/OperatedInTheNightlyBuildSystem
9 http://www.codesourcery.com/qmtest
10 http://www.gnu.org
11 http://www.fsf.org
12 http://cern.ch/atlas-computing/projects/qa/qa.php
13 http://cern.ch/atlas-computing/computing.php
14 https://twiki.cern.ch/twiki/bin/view/Atlas/WorkBook
15 http://www.stack.nl/~dimitri/doxygen/
16 http://cern.ch/atlas-php/docs/docs.php
17 http://twiki.org/
https://twiki.cern.ch/twiki/bin/view/Atlas/AtlasComputing
18 http://hepwww.ph.qmul.ac.uk/twiki2pdf/
19 N. McCubbin et al., Review of ATLAS Software Documentation, ATL-SOFT-2006-003,
http://documents.cern.ch/cgi-bin/setlink?base=atlnot&categ=PUB&id=soft-pub-2006-003

Figure 1: The software life cycle - different time scales for different types of releases.

Figure 2: The architecture of the Nightly Build and Test system (NICOS)

Figure 3: Information provided for each Nightly Build and Test.

Figure 1

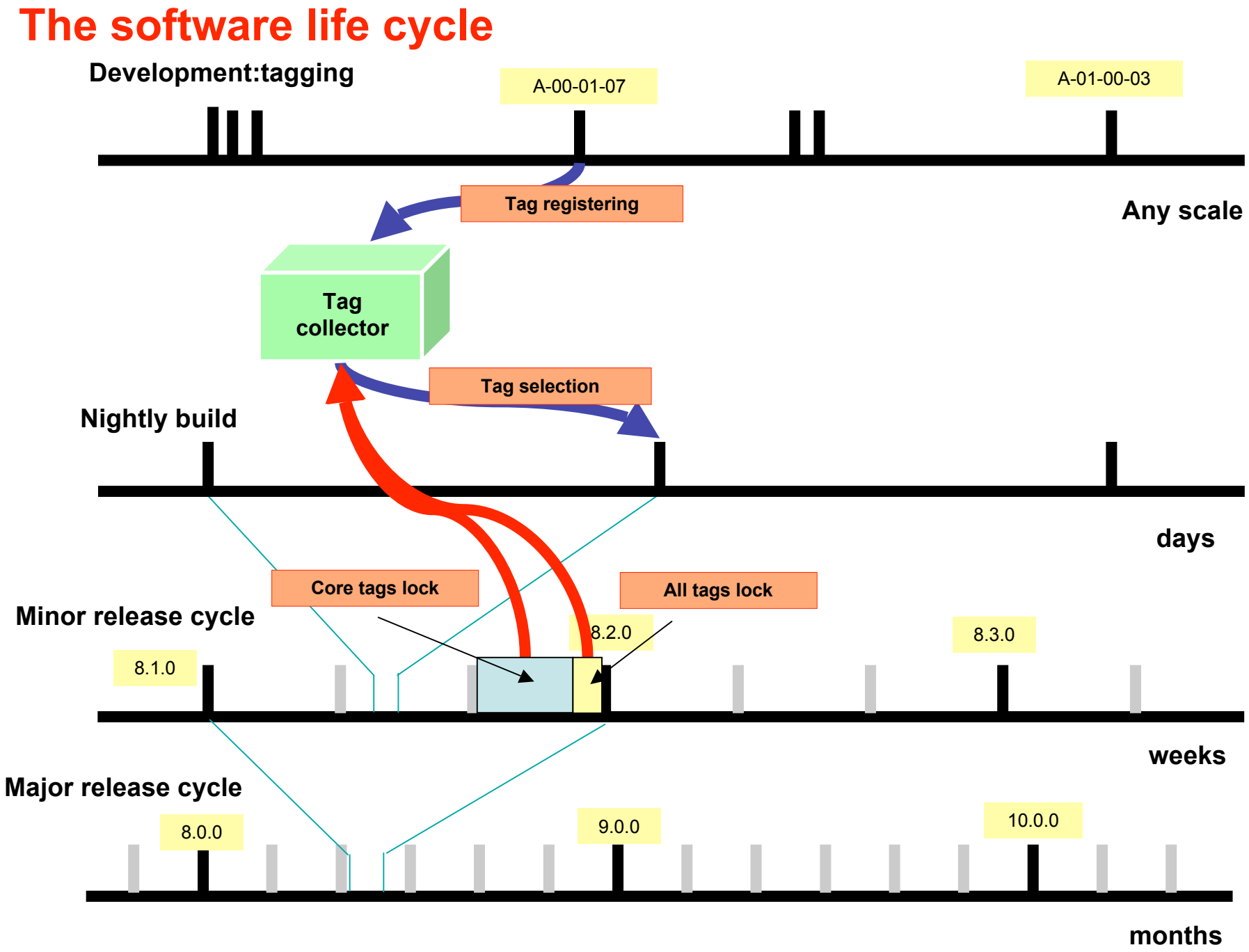

The software life cycle
Development:tagging
A-00-01-07
A-01-00-03
Tag registering
Any scale
Tag collector
Tag selection
Nightly build
days
Core tags lock
All tags lock
Minor release cycle
8.1.0
8.2.0
8.3.0
weeks
Major release cycle
8.0.0
9.0.0
10.0.0
months

Figure 2

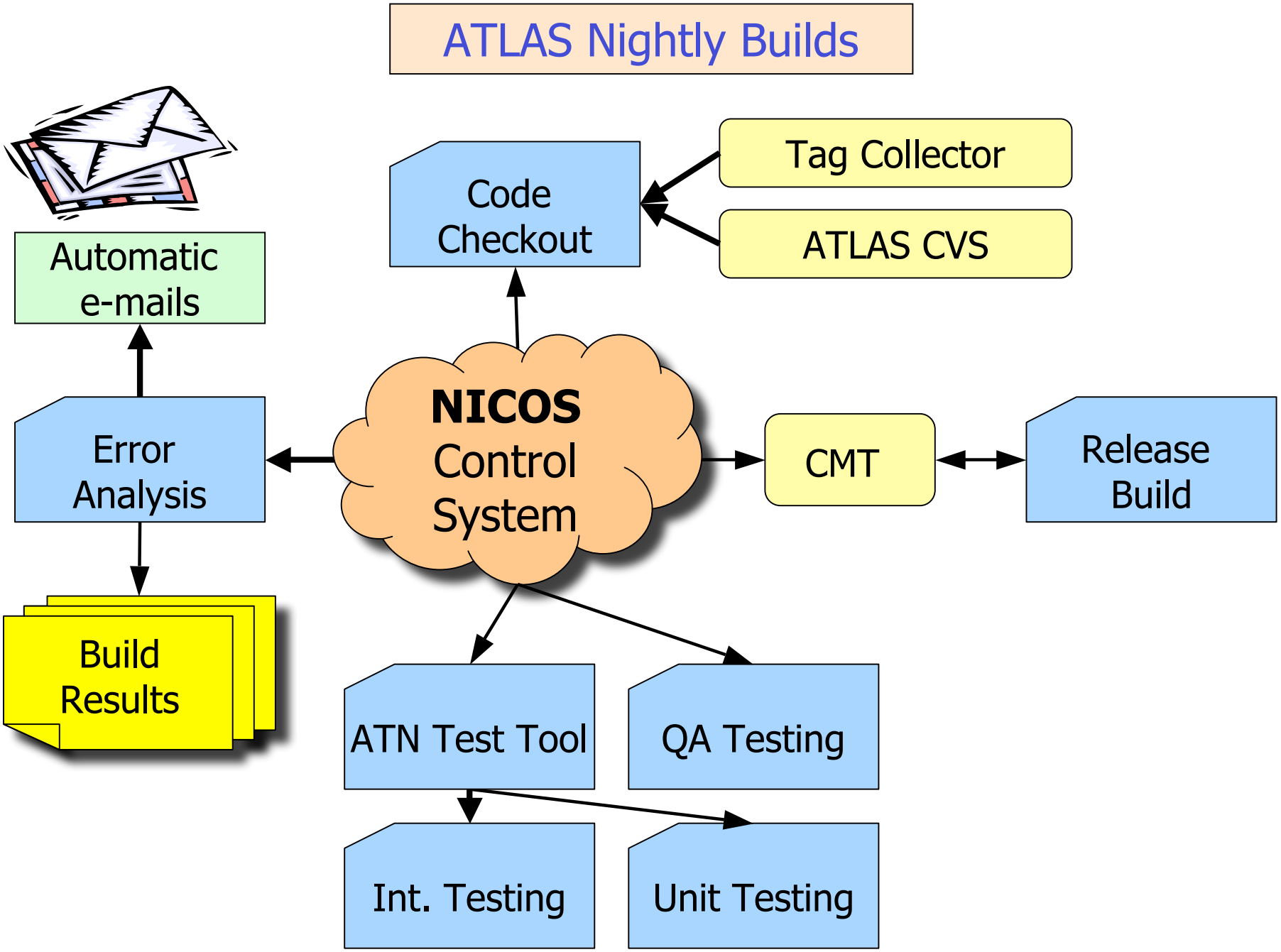
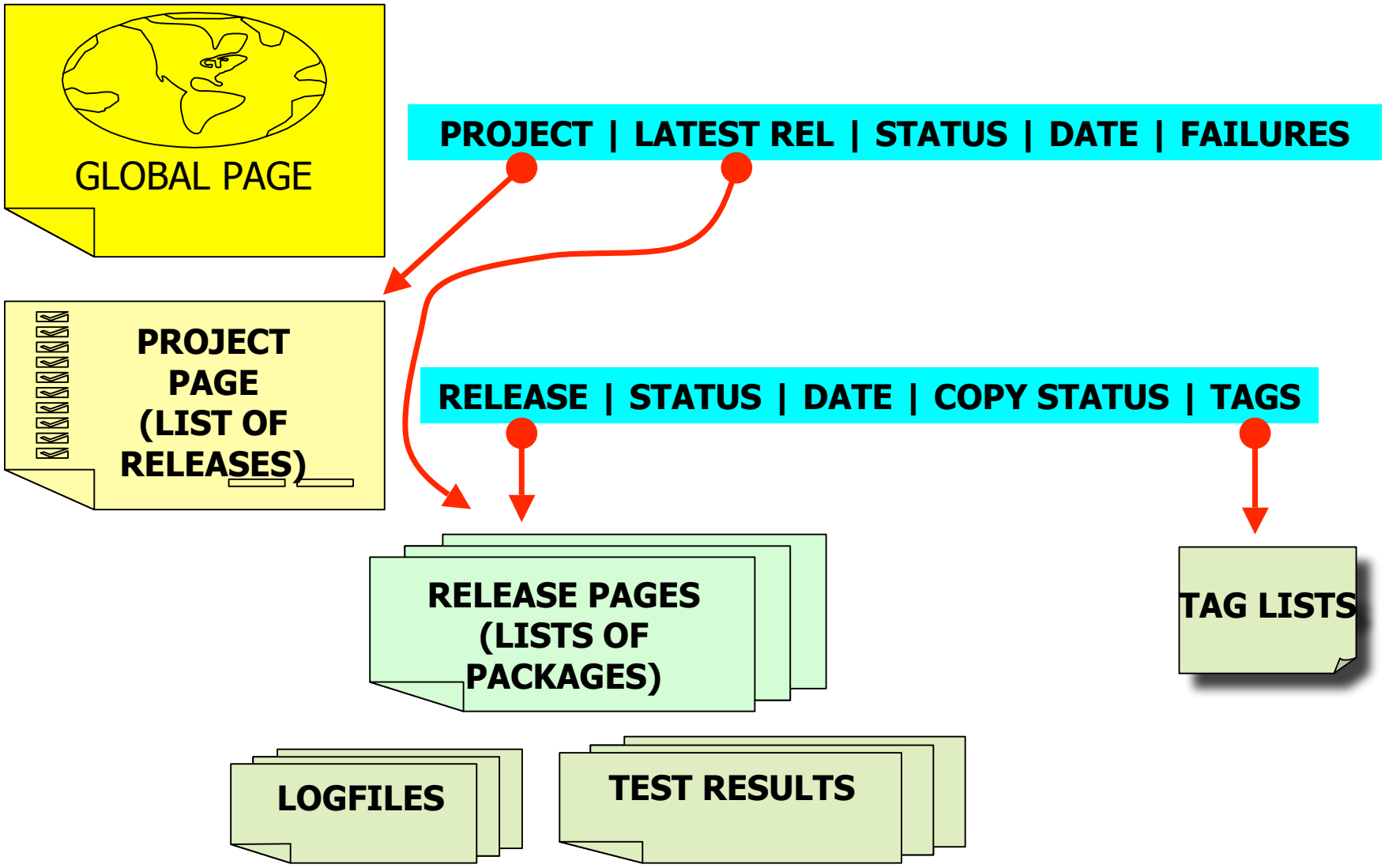

ATLAS Nightly Builds
Automatic e-mails
Code Checkout
Tag Collector
ATLAS CVS
Error Analysis
NICOS Control System
CMT
Release Build
Build Results
ATN Test Tool
QA Testing
Int. Testing
Unit Testing

Figure 3


GLOBAL PAGE
PROJECT | LATEST REL | STATUS | DATE | FAILURES
PROJECT PAGE (LIST OF RELEASES)
RELEASE | STATUS | DATE | COPY STATUS | TAGS
RELEASE PAGES (LISTS OF PACKAGES)
TAG LISTS
LOGFILES
TEST RESULTS